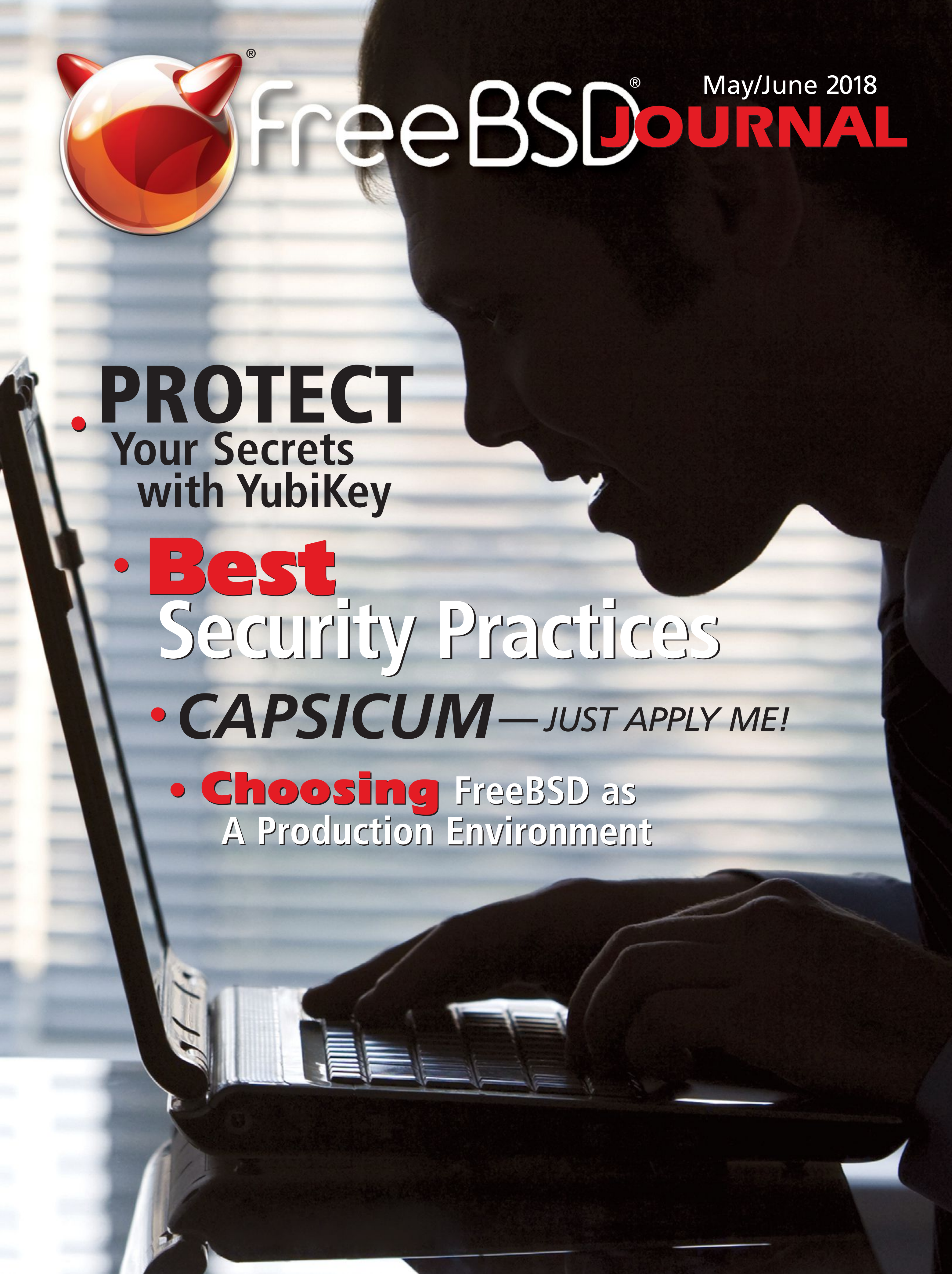
FreeBSD® JOURNAL

May/June 2018

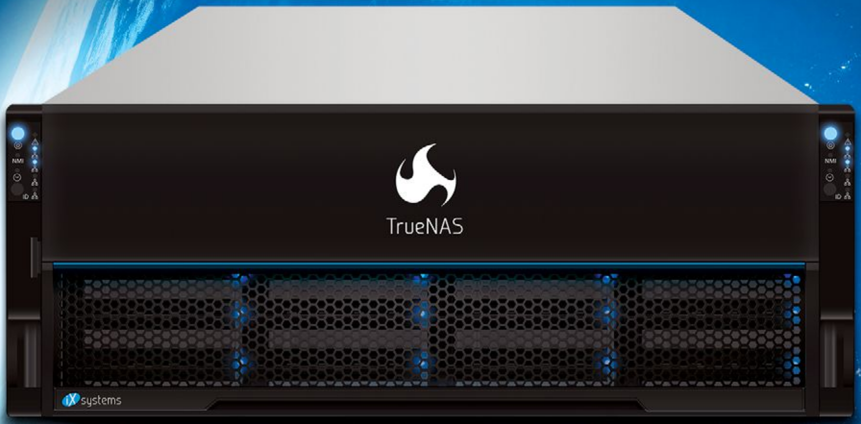


- **PROTECT** Your Secrets with YubiKey
- **Best** Security Practices
- ***CAPSICUM*** — *JUST APPLY ME!*
- **Choosing** FreeBSD as A Production Environment

Groundbreaking TrueNAS® M Series

intel XEON® SILVER inside™

DISRUPTING STORAGE AT WARP SPEED



LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit iXsystems.com/TrueNAS or call (855) GREP-4-iX today!

HGST a Western Digital brand

iX systems™

Table of Contents

May/June 2018

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™ FOUNDATION

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher	Walter Andrzejewski walter@freebsdjournal.com
Editor-at-Large	James Maurer jmaurer@freebsdjournal.com
Copy Editor	Annaliese Jakimides
Art Director	Dianne M. Kischitz dianne@freebsdjournal.com
Office Administrator	Michael Davis davism@freebsdjournal.com
Advertising Sales	Walter Andrzejewski walter@freebsdjournal.com Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

25TH ANNIVERSARY FreeBSD

Hello and welcome to the May/June 2018 issue of the *FreeBSD Journal*. Instead of the usual Foundation update, we wanted to take a minute to say ***Thank You!***

June 19, 2018 marks the 25th Anniversary of the FreeBSD Project. We couldn't have reached this historic milestone without you, the FreeBSD community. From its small beginnings at UC Berkeley to the respected open source operating system used by both large companies and individuals alike, your passion and dedication to the Project is responsible for its longevity and growth. We hope you'll join us in celebrating FreeBSD on June 19 and throughout the rest of the year.

FreeBSD FOUNDATION

SEE TEXT ONLY

BY MICHAEL W LUCAS

Best Security Practices

We're talking security. Everybody likes security.

Hackers! Theft! *Big wall displays showing the dotted line where the Bad Guys routed their traffic through country after country! One team, two team, red team, blue team! It gets your heart pumping without even needing to get up out of your chair.*

What nobody likes is doing all the things that prevent all the exciting hacker stuff. Applying patches and locking down all those fiddly little bits feels like wasting time. In reality, though, best practices save time. The energy you spend applying patches is trivial next to the energy you'll spend trying to scrape an intruder out of your servers. Best system administration practices, like diet and exercise, can feel flat-out tedious. Also, much like diet and exercise, few people can agree on what best practices are.

Here I present what I consider essential security practices for every host. In FreeBSD's case, figuring out what best practices are is complicated by decades of obsolete documentation. FreeBSD 12 obsoletes much of what was best practice in FreeBSD 6, and what we did with FreeBSD 1.1.2 is utterly irrelevant.

Your organization might impose additional practices. If you have a centralized logging server, an LDAP server for authentication, or an SSH certificate authority, use them. If you have an automation system such as Ansible, have it implement these practices across your environment.

Always start by listening to FreeBSD. It'll tell you most of what you need to know.

Status Mails

FreeBSD schedules automatic system checks every day, week, and month, using the periodic(8) command. The results of these commands get emailed to the system administrator. The simplest way to find out what's going on with a host is to regularly read those status mails.

The emails are sent to the root account on the local host. Many programs can email root when they go belly-up. It's best to go into /etc/aliases and redirect root's email to an address that someone actually reads, as shown here.

```
root: flunkies@mwl.io
```

Save the file and run `newaliases(8)`.

Most sysadmins I know have a long-standing goal of reducing the amount of email they receive. Yes, adding these messages impedes that goal. These emails have been collaboratively developed to contain the absolute minimum information necessary, however. You need notices such as "this package has a security vulnerability" and "your FreeBSD version just passed end-of-life and maybe you should upgrade while you can."

Additionally, you can adjust the checks these jobs perform. Maybe you have a rock-solid monitoring system that always catches when a partition or pool is about to fill, and you don't want the emails to contain filesystem utilization information. Disable and enable checks via `/etc/periodic.conf`. Like many other FreeBSD services, you'll find default settings in `/etc/defaults/periodic.conf`. By setting the task name to YES or NO in `periodic.conf`, you'll toggle that check.

Take a look through `/etc/defaults/periodic.conf` and the actual scripts under `/etc/periodic`. Some of the disabled jobs will be useful in your environment.

If you have a large staff, reading server status mails is a wonderful task to delegate to junior sysadmins. When I'm the senior sysadmin, I'll usually set the host to send all root mail to an alias on my mail server. That alias redirects all email to myself and my Trusted Lieutenant—or, rather, a lieutenant who thinks she's worthy of trust and swears she'll meticulously read every email and either resolve any issues or bring them to my attention. I can perform spot checks on those emails, watching for a host to report a problem. When my trusted lieutenant neither resolves nor reports the issue, I demote her to Lieutenant Formerly Known As Trusted and the real fun begins.

The pre-scheduled `periodic(8)` jobs don't cover everything you might need to schedule, however.

Update Checks

Once upon a time, the only way to apply security patches to FreeBSD was to build the operating system from source. Knowledgeable folks could avoid building the whole tree by building only the affected programs, but that's highly questionable with software like OpenSSL that has its grubby little fingers dang near everywhere. But the worst bit about applying security patches by hand wasn't building them; it was knowing that a patch was available and determining if the underlying problem affected your hosts.

The freebsd-update(8) program vastly simplifies security patches for the vast majority of FreeBSD users. Security patches and upgrades require only simple commands. Best of all, freebsd-update(8) tells you if a patch is available. Add a job to check for security updates to root's crontab.

```
1    1    *    *    *    freebsd-update cron
```

This tells freebsd-update(8) to wait for a random number of minutes then query FreeBSD's update servers for any new security patches for your version of FreeBSD. If it finds any new patches, it downloads them and emails root. Your Trusted Lieutenant has another chance to notice messages from your hosts.

When you notice a new set of security patches, go check out the corresponding security advisory. There's always one. See how badly this problem affects you. Do you need to apply patches over lunch, or will they wait for after hours or maintenance day? Or do you need to re-issue all your TLS security certificates because of OpenSSL… again?

Once you know how bad the problem is, and how immediately it affects you, you can apply those patches. Verify that you have a good backup. If you're using ZFS, create a new boot environment so you can easily revert. (While I've never had a freebsd-update(8) security patch go badly, I've been a sysadmin too long to not prepare a fall-back path) Now you can apply your patches.

```
# freebsd-update install
```

You'll be prompted to reboot. Depending on what gets patched, freebsd-update might tell you to reboot and run freebsd-install once more. Follow its instructions.

The freebsd-update(8) check also notifies you if your FreeBSD version has passed its End Of Life date. That makes it vitally important you upgrade to a newer version. Freebsd-update(8) makes updating to a newer release easy, but it won't do any downloads for you. It doesn't know which release you'll want to upgrade to. Are you going to run FreeBSD 11.5, or are you jumping to 12.1? That's a decision only you can make. Pick a version and use the `-r` flag to specify it.

```
# freebsd-update upgrade -r 12.1-RELEASE
```

Annoyingly, the release name must appear in the same case as the official release name. It's not 12.1-release, it's 12.1-RELEASE. You'll be prompted to compare a few critical configuration files that you've changed on your host. Decide if you want to keep or discard the changes. The program downloads the updates, and then you'll need to install them with another **`freebsd-update install`** command. For a full version upgrade, you'll certainly need to reboot and run the install command again.

If you're building FreeBSD from source rather than using freebsd-update, that's fine. I'm sure you have your reasons, and they might—*might*—even be valid ones. Just make sure that your environment is set up to distribute the upgrade across your network as quickly as possible. That's most often done by building on one centralized host and letting the other servers NFS mount `/usr/src` and `/usr/obj` for speedy installations.

Once you've upgraded the operating system, consider your packages.

Packages

FreeBSD's base system is deliberately small, compared to many other open source operating systems. It doesn't ship with a modern graphical environment, an SQL database, or even a web server. All those functions come from add-on packages. While those packages are easy to install and maintain, they require the same devoted, loving attention the base system needs.

FreeBSD's packaging system includes a tool to check for known security vulnerabilities in installed packages, pkg-audit(8). This tool gets run every day as part of the daily status checks, but those runs only provide the name of packages that have known security vulnerabilities. Running the tool on its own highlights the package problems. You can have your automation system run pkg-audit(8) across all of your hosts to get a master list of all vulnerable packages.

```
# pkg audit
python27-2.7.14_1 is vulnerable:
python 2.7 -- multiple vulnerabilities
CVE: CVE-2018-1061
CVE: CVE-2018-1060
CVE: CVE-2017-9233
CVE: CVE-2016-9063
CVE: CVE-2016-4472
CVE: CVE-2016-0718
CVE: CVE-2012-0876
WWW: https://vuxml.FreeBSD.org/
freebsd/8719b935-8bae-41ad-92ba-
3c826f651219.html
…
5 problem(s) in the installed packages found.
```

We have a few packages with security problems. The CVE identifiers let you look up the exact flaw, what it affects. You could use this information to determine if any of these vulnerabilities can affect your environment. The easiest fix is to see if FreeBSD has an upgraded package available with **`pkg upgrade`**. If you're using ZFS, create a boot environment before upgrading your packages. This will let you easily revert any changes.

```
# pkg upgrade
Updating FreeBSD repository catalogue...
Fetching meta.txz: 100%     944 B   0.9kB/s    00:01
Fetching packagesite.txz: 100%    6 MiB   6.4MB/s    00:01
…
The following 2 package(s) will be affected (of 0 checked):

Installed packages to be UPGRADED:
       perl5: 5.26.1 -> 5.26.2
       freetype2: 2.8_1 -> 2.8_2

Number of packages to be upgraded: 2

14 MiB to be downloaded.

Proceed with this action? [y/N]: y
```

The package manager will download and install the latest packages. You can then run `pkg audit` again to see which packages remain. In this case, python27 shows up again. The package upgrade remediated the installed Perl and freetype2, but didn't change python. And looking at all those CVE numbers in python27, that's a pretty long list of problems.

How can one program have so many problems?

Because nobody's fixed them, obviously.

Maybe the problem comes from the original package. Python 2.7 might have known security problems, but the python authors might have decided that eliminating those problems would unacceptably change how python behaves. In some software, the software author might dispute that a security issue needs fixing. In any case, it's better that you know the problem exists.

Perhaps the software vendor has created a fix, but the FreeBSD port isn't yet updated. Maybe the port is updated, but the new package isn't yet built and distributed to the mirrors. Maybe the fixed package is available in the bleeding edge packages, but not in the quarterly branch deployed by default.

If the problem affects you critically, look to see where the fix is held up. Open source software is community-maintained. This is your chance to contribute. If you can't fix the problem, you can at least determine how long it will be until a fix is available. If it won't ever be fixed, you can make plans to mitigate your risks.

Packet Filtering

I strongly recommend running a packet filter on all hosts. Even a simple packet filter that says "all outbound connections are allowed, but only these inbound connections" will help protect you against rogue software. Don't rely on your network firewall to block all malicious traffic; if someone weasels their way into the network, they might attack your host from another machine. While FreeBSD has three different firewall suites, my informal surveys show that 80% of FreeBSD sysadmins prefer pf(4), so we'll use that.

Start by creating a simple firewall configuration in `/etc/pf.conf`.

```
ext_if="em0"
set skip on lo
scrub in
block in
pass out
pass in on $ext_if proto tcp to ($ext_if) port {22, 53, 80, 443}
pass in on $ext_if proto udp to ($ext_if) port {53, 33433 >< 33626}
pass in on $ext_if proto icmp
```

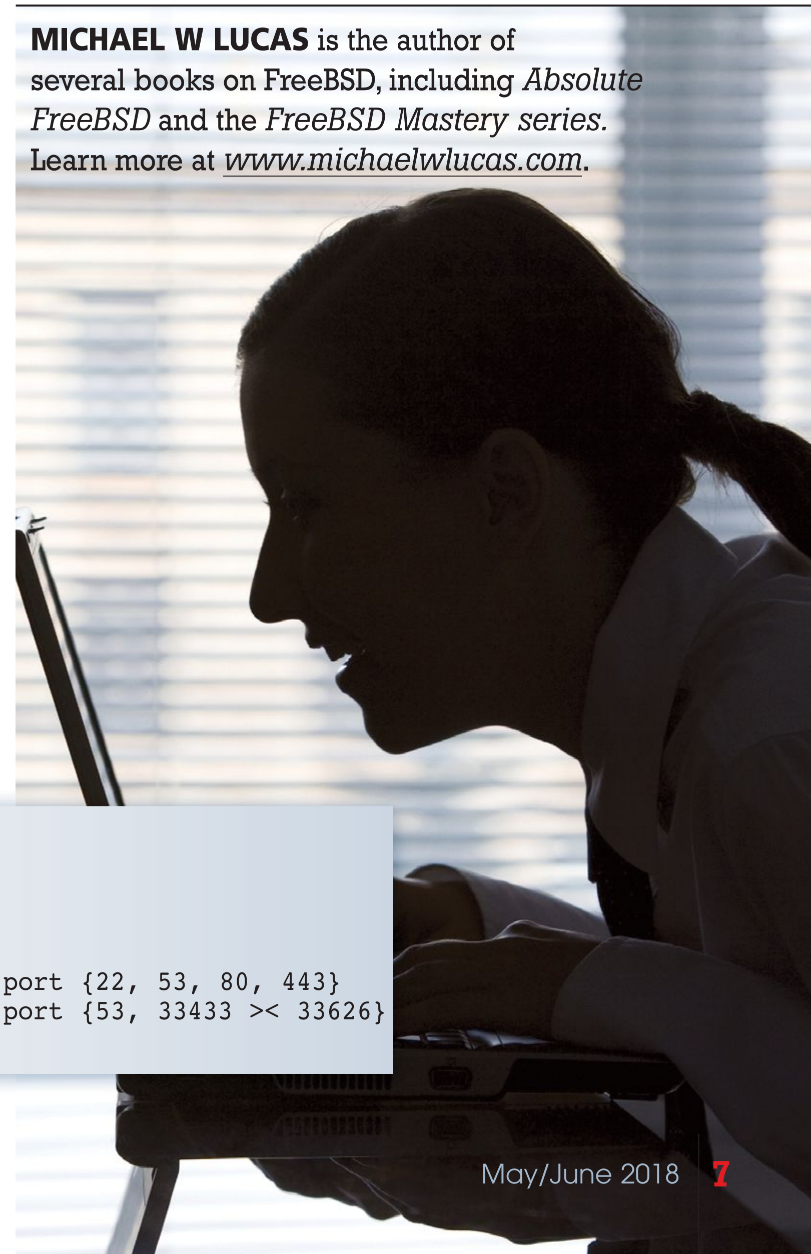
This ruleset disallows all inbound traffic but permits outgoing traffic. We then allow connections on four TCP ports: 22 (SSH), 53 (DNS), 80 (HTTP) and 443 (HTTPS). We allow a few more UDP ports: 53 for DNS, and 33433 through 33626 for traceroute. Finally, we allow inbound ICMP.

If your host doesn't run DNS, remove the TCP and UDP references to port 53. If you don't have a web server, you can remove TCP ports 80 and 443. You probably want traceroute and ping, for diagnostic purposes if nothing else.

Enable PF with the /etc/rc.conf entry `pf_enable=YES`, then start it with `service pf start.`

I could go on at length about unprivileged users and securelevels and vulnerability assessment, but you'll find extensive information about all of these in any systems administration book. If you start your FreeBSD systems here, though, you'll have a good start. ●

MICHAEL W LUCAS is the author of several books on FreeBSD, including *Absolute FreeBSD* and the *FreeBSD Mastery series.* Learn more at www.michaelwlucas.com.



SEE TEXT ONLY

Protect Your Secrets

Do you create complicated passwords? How hard is it for you to remember the new ones? Do you keep your credentials on a Post-it note near your monitor? If you do any or all of the above, we have an alternative that will help protect your privacy and make it easier for you to stay safe.

YubiKey Overview

A lot has been said about YubiKey, which is produced by Yubico. It has become one of the most popular solutions offering a secure 2FA — Two-Factor Authentication. It also may be used as a secondary password factor U2F — Universal 2nd Factor. It offers strong authentication and is easy to use.

YubiKey is a USB-like device. In the simplest use-case, when we connect YubiKey to a computer, it is detected as a keyboard in the operating system and the device allows us to store up to two passwords. Depending on how long we press the button on the device, it will release the first or the second password. In this simple use-case, we can use it to integrate with almost any web service.

In this scenario, we can use it to remember our passwords and forget about using Post-it notes. For more advanced users, this scenario can help protect their data. Rotating passwords is difficult, as it is problematic to remember new ones. Now, we can rotate our most often used key (for example access to our password vault) and not have to bother remembering it (although it is always a good idea to have backup). While keeping our primary password on the YubiKey device, we can still use other devices (such as a mobile phone) as a second factor.

By Jarosław Zurek, Michał Borysiak, and Mariusz Zaborski

Authentication Methods

Besides storing static passwords, some models of YubiKey also support more sophisticated authentication methods such as:

• **One-time Password (OTP)** – authentication mechanism, generating passwords that can be used once.
• **OATH – HOTP** – event token, generating 6- or 8-character OTP passwords using the HOTP algorithm.
• **OATH – TOTP** – 6- or 8-character OTP passwords, the TOTP algorithm is based on time function.
• **PIV (Personal Identity and Verification)-Compatible Smart Card** – enables the use of private RSA/ECC keys stored on YubiKey for signing and decryption. This mode works like a smart card.
• **OpenPGP** – encryption and signing using RSA and ECC private key, stored on YubiKey, using standard suits like PKCS #11.
• **U2F** – an open authentication standard that enables secure access to any number of online services. Only one single device, without additional drivers, or client software.

It is possible to use two-step verification with web services like Google, Facebook, GitHub, and Hotmail. This is useful because even if attackers steal the first factor of authentication (username and password to the account), they are not able to log in. We can ensure this by using a YubiKey device which supports U2F. Of course, we must enable two step verification on a chosen web service.

Models

There are a few different types of YubiKeys that can be used for various purposes. The most basic version is the *FIDO U2F Security Key*. It supports static password authentication and may be integrated with the most popular applications like Gmail and Facebook using U2F. It does not support methods like HOTP or OpenPGP. This device is the most affordable product they offer at the time of writing this article and the cost is US $18.

More advanced models, known as the YubiKey 4 generation (which also includes YubiKey 4C, YubiKey Nano and YubiKey 4C Nano) already support basic cryptographic methods like OTP or OATH modes. Differences are in device size and USB port type. Their use is definitely wider than the previously mentioned model and some examples will be explained in detail later in the article.

Another available option is YubiKey NEO. An additional feature supported by this model is communication via NFC. For example, after tapping the device, a smartphone can read OTP emitted by YubiKey. A summary of differences can be seen in Table 1.

	FIDO U2F Security Key	YubiKey 4 generation	YubiKey NEO
OTP		✔	✔
OATH – HOTP		✔	✔
OATH – TOTP		✔*	✔
OpenPGP		✔	✔
U2F	✔	✔	✔
Secure Element	✔	✔	✔
Smart Card (PIV)		✔	✔
Supports NFC communication			✔

**Requires additional app (lack of built-in Real Time Clock).*

Table 1: Comparisons of different YubiKey models

FreeBSD Tooling

YubiKeys come with a set of open source tools that are necessary to integrate YubiKey in a Unix-like environment. For FreeBSD users most of these tools are available in the ports collection as well as in the binary package repository. Two of the most interesting packages are `security/ykpers` and `security/yubico-pam`. The `security/ykpers` package contains several command line tools to manage YubiKey:

• `ykpersonalize(1)` —is necessary to program YubiKey; it handles configuration options for almost every YubiKey feature.
• `ykinfo(1)` —is useful for retrieval of basic information about YubiKey.
• `ykchalresp(1)` —allows for signing data using YubiKey in a challenge-response mode of operation.

GELI Full Disk Encryption

GELI is the most popular disk encryption method on FreeBSD. One of the most secure ways of using GELI and FreeBSD is to use two-factor authentication with a passphrase and a key file. The key file is kept on a memstick. When we boot our machine, we need to provide both factors. After decrypting our device, we remove the memstick with the file. In that way, if somebody steals our computer they will also need to see our password and steal our memstick. If we want to be even more paranoid, we can keep a kernel and a key file on the memstick. In that way, even if somebody has access to our computer, it makes it impossible to integrate with our software. An intruder could still alter our hardware, but this is much harder.

FreeBSD has recently begun supporting full disk encryption, which means that even a kernel is encrypted—only the small boot loader partition is not. Unfortunately, in the current implementation, we do not support a key file, so we can only use a passphrase to encrypt our disk. Thanks to YubiKey, which can be detected as a simple keyboard, we can use it to provide a passphrase during boot. Using it in that way, we lose one factor. We can mitigate this by using a passphrase which we provide using a normal keyboard and a second part of the password which is much longer and is kept on YubiKey. In that scenario, somebody not only would need to see what passphrase we are typing, but also would need to steal our YubiKey.

To make the device to meet these requirements, we program the second slot of Yubico in static mode:

```
$ ykpersonalize -2 -o static-flag -o append-cr
```

Thanks to the **append-cr** flag we do not have to press Enter after each use of this slot. In the case of GELI, we would first provide our passphrase and then the second factor using YubiKey. By default, there is no output when typing a password in GELI. In our case, we would see that the passphrase was submitted because the factor provided by YubiKey would contain trailing ENTER.

Integration with FreeBSD Login

Yubico provides a PAM module which can be deployed within existing authentication systems. The module can be found in the **security/pam_yubico** package and it works in two modes: online and offline authentication. The former provides a second factor based on one-time passwords, but it delegates authentication to Yubico cloud services. Therefore, it requires a stable internet connection. On the other hand, it is very easy to setup a newly bought Yubico as a second authentication factor for your system account.

Every device has a secret on its first slot preset by the manufacturer. The slot works in so-called Yubico OTP mode. Each password generated from this slot consists of a static 12-character part and the remaining dynamic part. The static part never changes and it can be considered the device's public identifier. These factory defaults are already known by Yubico cloud services. All that we need to do is to retrieve the API token and the user ID using a special form available on Yubico's website.

The second mode of operation is more practical. It requires a YubiKey to work in a challenge-response mode when the device can be issued to sign data sent by a user application. In this particular example, our application is the Yubico PAM module.

First, we have to program the device to work in a challenge-response mode. In the example below, we will use the first slot:

```
$ ykpersonalize -1 -o chal-resp -o chal-btn-trig -o chal-hmac -o hmac-lt64 -o serial-api-visible
```

We want the device to wait for a user confirmation of each operation which is guaranteed by the **chal-btn-trig** flag. The user has to press the key located on the YubiKey while logging into the system, unfortunately twice. The first press is needed to check whether the challenge located in a file matches the device response. If so, the second press generates a new challenge-response pair and stores it for later use. We need to generate an initial challenge which is stored by default in **~/.yubico** directory:

```
$ ykpamcfg -1 -v
```

The next step is to configure the PAM module. We want to use the second factor together with a static password to protect any login to our computer. For this purpose, we will modify the **/etc/pam.d/system** file so the "auth" section will look as follows:

```
# auth
auth    required    pam_unix.so         no_warn try_first_pass
auth    required    /usr/local/lib/security/pam_yubico.so mode=challenge-response
```

Now issue the `sudo -s` command, type the static password and press the button on the device. It waits for a user action up to 15 seconds and an LED indicator is blinking during this time. Authentication will fail either when the user does not take action or if YubiKey is not connected to the USB port.

Integration with SSH

Another useful application of YubiKey is strengthened remote authentication to an SSH service. A commonly described method utilizes the Yubico PAM module using the online mode of operation. However, a Yubico token is compliant with OATH-HOTP standards and, therefore, it can work with any authentication server which supports this standard. For example, Wheel Cerb AS is such a multi-factor user authentication solution. We will use the `pam_oath` module which is included in the `security/oath-toolkit` package and does not require a connection to any external cloud service.

We need to reprogram our YubiKey to support OATH mode. Unfortunately, we have only two slots on our device so we need to overwrite one of the previous configurations:

```
$ ykpersonalize -1 -o oath-hotp -o oath-hotp8 -o append-cr
```

We want to use 8-digit long, counter-based passwords. The output of the command should contain a line with a key in hexadecimal form:

```
      …
key: h:c621245c5f05eefec1d9f2960f34b865849dd074
  …
```

We need to store the user's name and the key in the pam_oath database file on the target machine (of course "alice" and the key should be replaced by real values):

```
$ echo "HOTP alice - c621245c5f05eefec1d9f2960f34b865849dd074" >> /usr/
local/etc/users.oath
```

The next step is to modify the sshd PAM configuration in order to enable the pam_oath module. We can achieve this by modifying the /etc/pam.d/sshd file so the "auth" section will look as follows:

```
 # auth
 …
 auth        required      pam_unix.so         no_warn    try_first_pass
 auth        required      /usr/local/lib/security/pam_oath.so usersfile=/
                           usr/local/etc/users.oath window=16 digits=8
```

We need to ensure the sshd configuration which is placed in the /etc/ssh/sshd_config file contains a few options set to the appropriate values as follows:

```
ChallengeResponseAuthentication yes
PasswordAuthentication no
UsePAM yes
```

Finally, reload sshd(8) service and try to login to the remote server typing the static password and then using the dynamic password from the token.

Conclusion

YubiKey is a remarkable device that can be used in a corporation or by individuals to increase their security. This small device supports many different authentication methods and can be used with many popular web services as well as with programs like SSH. It also allows us to leverage some of the imperfections of tools that do not support a 2FA. It can be used as a second factor or to keep the primary password. It is an interesting alternative to other solutions like mobile applications. YubiKey also allows us to painlessly change our passwords without the need for any memorization. ●

JAROSLAW ZUREK is a software developer at *Wheel Systems* where he supports a project creating a privileged session manager. He is interested in cryptography, TLS, and low-level/hardware programming.

MICHAL BORYSIAK is a software developer at *Wheel Systems,* where he works on centralized authentication systems. He is fascinated by low-level operating system concepts, networks and cybersecurity.

MARIUSZ ZABORSKI is a lead software developer at *Wheel Systems*. He has been a proud owner of the FreeBSD commit bit since 2015. Mariusz's main areas of interest are OS security and low-level programming. At *Wheel Systems*, Mariusz leads a team that is developing the most advanced solution to monitor, record and control traffic in an IT infrastructure. In his free time, he enjoys blogging (http://oshogbo.vexillium.org).

BSD CERTIFICATION GROUP HAS JOINED WITH LPI.

ONE MORE STEP TOWARDS A FREE AND OPEN SOURCE WORLD

NOT TO MENTION, JOBS!

Now as a part of LPI, BSD certification will gain a new global reach. It will also benefit as it's relaunched in 2018 to fit into a broader program of free and open source professional credentials. You can help by participating in retooling the certification at lpi.org/bsd.

Linux Professional Institute

COMING IN 2018. WATCH FOR UPDATES.

SEE TEXT ONLY

Choosing FreeBSD AS A PRODUCTION ENVIRONMENT

BY ASHIK SALAHUDEEN

I lead a small team of developers who write and operate software that serves a small (but growing) clientele at Accordium APS. The following is an account of how we ended up choosing FreeBSD as a production environment and some of the pros and cons of that choice.

HOW DID WE END UP WITH FreeBSD IN THE FIRST PLACE?

I have been a Free and Open Source Software advocate since 2001 and have been using Linux machines as primary drivers ever since. Until 2017, my only close encounter with FreeBSD was when I set up an LDAP instance sometime in 2007. I thought it was sane then, but never gave it much thought. 2017 was also the year when Accordium came into being, and the first time I got to be fully in charge of a production environment right from the get go.

Doubting Linux-based Operating Systems–systemd

Around 2014, the Linux ecosystem started undergoing a change that I didn't understand at first—when the systemd project started making news. I didn't give it much thought then, but later on, when Arch Linux (my preferred OS then) started pushing me towards it, I had to start using it. Immediately, I encountered a bunch of warts that I would, in hindsight, describe as growing pains.

My operating systems pre-systemd were a kernel and a bunch of programs, very loosely coupled, and while I hated the lack of uniformity, I at least knew what my computers were doing most of the time. At this point, I was getting exasperated with the general state of things, especially when most mainstream distributions decided that they'd just force systemd on everyone (notable exception: Gentoo).

I tried to live with it for a while. But it constantly broke my expectation of how my computer would run. The changes were pervasive and no amount of grappling with it would give me an idea of just what it set out to accomplish. Later on, I realized that systemd was trying to present a coherent interface--unify all Linux based operating systems—which is a laudable goal.

My computers would constantly refuse to shut down, with cryptic messages like "A stop job is running for <something>" and then hanging in there. They would start-up with cryptic error messages and troubleshooting them made me an expert in trawling github and the Unix stackoverflow. Coupled with the constant stream of vulnerabilities and the project's ridiculous responses to it, I really wanted to switch from using operating systems that used systemd.

Evaluating BSDs

In November 2016, I started to consider BSDs as a potential refuge, at least for personal use. I have been friends with Cherry G. Mathew, a NetBSD developer, for a long time, and was toying with the idea of using a BSD, at least for personal use. He, of course, stood up for NetBSD and tried to get me to use it, but also very helpfully told me what it was not capable of doing—like running VirtualBox. I decided to look for a systemd-free Linux and stick with it on my desktop.

Around the same time, Philip Paeps (trouble) spent a few days with me, and in one of our chats, he explained the coherence of FreeBSD—and the fact that it was a single operating system developed along with applications. He also told me about FreeBSD's user base on the server side, and I was surprised to learn that it was a very popular choice.

Of course, I set up a VM and tried it out to see how a Linux user would fare in a FreeBSD environment. I was productive from day one, and for most things I couldn't figure out, I just had to read the handbook.

FreeBSD AT ACCORDIUM

Accordium's development officially started in April 2017 with just two developers (including me). I found myself in charge of selecting the server environment and with nobody to whom I had to justify it. This was the first time I had to make choices regarding the entire software stack, and I was keen to at least test out FreeBSD. To my relief, there was a FreeBSD image available for deployment on Amazon Web Services.

Our requirements were very minimal, just a few API end points written in Spring Boot framework, running on JVM, a Postgres database server and a rabbitmq instance—all of which were available as binary packages. At the time, there was no production environment and the only users were us.

All of our development happened in Linux machines. Ahmed Uways Zulkurnain, the engineer who manages all our production machines now, was unfamiliar with FreeBSD at the time. He raised

Uways is the first engineer to work at Accordium and has the following comments on FreeBSD:

“Prior to working at Accordium I was not familiar with any operating system other than Debian GNU/Linux. The only things I knew about BSDs were their legendary uptime and more permissive license. Despite my inexperience, I was able to learn how to install/configure applications, write startup scripts and configure firewalls without too much effort. I don't recall ever having to reference any documentation outside of the books and articles on the FreeBSD website. I was pleasantly surprised at how up-to-date the package versions were in the current production release. There aren't many things I have to do when it comes to managing our FreeBSD instances.

1. Keep up to date with security advisories.
2. Update packages when necessary.
3. Stay on the currently supported production release.
4. Periodically audit the logs.

One year later, FreeBSD has become my server operating system of choice. systemd logging has always annoyed me; I'm happy to once again have plain text log files.”

Ahmad Uways Zulkurnain discovered open source software in 2005. He has since explored a wide range of topics related to computing, primarily in the realm of information security. He is currently a full stack software developer at Accordium. He enjoys twisty puzzles and chess.

a couple of very logical questions, the chief one being, "Why not Linux, we are using it ourselves after all." I asked him to bear with me and test it out first, promising to change the environments if he ever found FreeBSD lacking. It is April 2018, and he is yet to come back with an objection.

I switched to using TrueOS in June 2017, so that there was at least one person using FreeBSD to develop. My development pace was not affected in any way—if it was not available via pkg, I had it in ports.

FreeBSD IN PRODUCTION—ACCORDIUM GOES BETA

Early December 2017 saw us releasing a limited beta of our software—a bunch of APIs running on JVM, fronted by a client application written in javascript. We just went with FreeBSD in production, without stopping to think about it—why change it when it works perfectly well?

Here are the things that made me happy I selected FreeBSD.

- Our computers never crashed because of operating system problems.
- Everything works as expected—log files in `/var/log`, configurations in `/usr/local/etc` or `/etc` and so on. It is all organized well.
- The service scripts for our applications were easy to make.
- The operating system configuration is centralized. The base system configurations go in `/etc/` and the local installation specific configurations go in `/usr/local/etc`.
- Software availability—not once did we have to scratch our heads saying,"This is not available in FreeBSD."
- Updates are regular and easy to make.

We've been running for about five months, serving an increasing number of customer needs. None of our problems were due to FreeBSD. It has stayed firmly out of the way, a tireless and dependable work horse.

WHAT DO WE WANT FROM FreeBSD NOW?

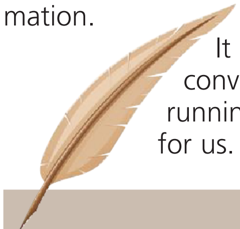
Accordium is a Java and Javascript shop. OpenJDK 8 works well on FreeBSD, but the next versions of Java (9/10/11) show no indication of supporting FreeBSD. The BSD port efforts seem to be minimal. When OpenJDK 11 rolls around, if there is no FreeBSD version, we'd be forced to move on, especially when interesting things like GraalVM are showing up.

WOULD WE RECOMMEND FreeBSD TO A START-UP?

If your applications are all one process per VM, FreeBSD is a safe, sane choice. I encourage everyone to give it a chance and save yourself a lot of pain.

If you need process/environment isolation (like Docker provides), FreeBSD has a stable, well tested feature called "jail" that would help you achieve it easily.

If you find yourself looking at OS-specific problems, the handbook will often have enough information.



It is mostly due to my personal convictions, but we found that running BSD machines made sense for us. ●

ASHIK SALAHUDEEN has been working computers for about 18 years, most of it with Unix-like operating systems and open source tech. He runs the engineering team at Accordium and works with other FOSS communities during spare time, primarily the indic language computing group, Swathanthra Malayalam Computing.

Testers, Systems Administrators, Authors, Advocates, *and of course* **Programmers** *to join any of our diverse teams.*

COME JOIN THE PROJECT THAT MAKES THE INTERNET GO!

★ DOWNLOAD OUR SOFTWARE ★

http://www.freebsd.org/where.html

★ JOIN OUR MAILING LISTS ★

http://www.freebsd.org/community/mailinglists.html?

★ ATTEND A CONFERENCE ★

BSDCan • June 6–9 • Ottawa, Canada
USENIX Annual Technical Conference • July 11–13 • Boston, MA
OSCON 2018 • July 16–19 • Portland, OR

The FreeBSD Project

FreeBSD FOUNDATION

SEE TEXT ONLY

By Mariusz Zaborski

Capsicum JUST APPLY ME!

Applying sandboxing always seems challenging and there is a historic reason for this. For example, using linux `seccomp(2)`–one of the oldest sandboxing techniques–requires a lot of effort. The simplest mode of `seccomp(2)` — *SECCOMP_SET_MODE_STRIC*– restricts a program to an unusable state. On the other hand, the most popular mode–`SECCOMP_SET_MODE_FILTER`–is very hard to apply and maintain and it can also easily betray us. In this article, we discuss a sandboxing technique built in FreeBSD called Capsicum. If the reader is interested in a deeper comparison between different popular sandboxing techniques, the author recommends a few articles [1] [2] [3]. Here we will discuss what Capsicum is, the tooling we have and, most importantly, how we can use it in our applications.

Overview of Capsicum

The Capsicum infrastructure can be divided into two parts:

- Tight sandboxing
- Capability rights

Tight sandboxing means we don't have access to any global namespaces. With a global namespace, we are referring to a limited area within an operating system. These areas have a set of names that allows the unambiguous identification of an object [4]. To simplify it, we can't operate on objects using their identifier like a filepath. This tight sandbox still allows us to operate with objects by using their handlers. In UNIX-like operating systems, we have one universal handler—descriptor—and we can operate on the file using a file descriptor. The same with a process. In this mode, we can't operate on a process using PID, but we can operate on it using a process descriptor [5]. In Capisucm, this mode is called Capability mode. In FreeBSD, we have one simple syscall to enter this mode—`cap_enter(2)`.

With Capsicum we go one step further by permitting limitations on the descriptors. If we have a descriptor that we know will be used only to read, we can set a specific right (`CAP_READ`) that will ensure this particular descriptor will be read-only. If there is an attempt to write on this particular descriptor, it will fail. We refer to these limitations as capability rights. It is always possible to limit descriptors further, but we can't extend the capability rights of a given descriptor. That means if we have a descriptor with the `CAP_READ` and `CAP_WRITE` capability and at some point we decide we don't want to read any more of this descriptor, we can drop the capability. For obvious reasons, it doesn't allow you to extend them. In FreeBSD, we have a special function—`cap_rights_limit(2)`—which allows us to limit descriptors. Currently we have 79 rights that allow granularity limit handlers.

Thanks to capability mode and capability rights we can ensure that a process has access only to objects that it really needs. This eliminates the ambient authority problem, where any process has access to all user data. If an attacker would exploit a tool like `grep(1)`, `patch(1)` or even `cat(1)`, he would gain access to all user data. An attacker could also create a new connection to an arbitrary server and send those files to it. In a Capsicum world, even if an attacker could exploit any of those tools, he would have read-only access to a few files on the disk. He wouldn't be able to overwrite any important data or send them over to the network.

Extending Capabilities

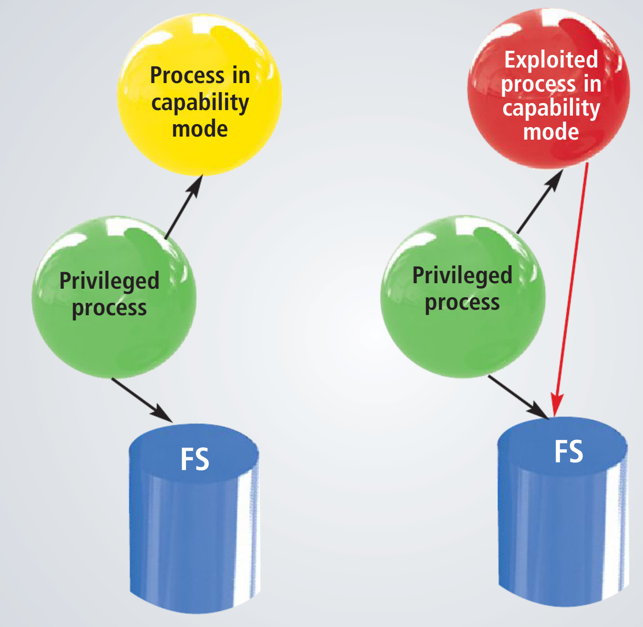
There are two methods by which we can obtain capabilities:

- initialization phase
- obtain them from another process

In the first case, we pre-open all connections, files etc. needed in our application before entering capability mode and we can only operate on those capabilities. This method is ideal for small applications that have limited functionality.

The second method is to use another process that already has the capability rights to the object we want. Because all objects are represented by descriptors, we can easily pass them from one process to another using UNIX domain sockets. If one process has a descriptor to talk with a server, it can pass it to another process, in which case both processes can operate on the single descriptor.

A very commonly used pattern with Capsicum is having two processes. The first has ambient authority so it can operate on behalf of the user, and the second is sandboxed. The ambient authority process is used only for simple things like opening a list of files or connecting to a server. The sandboxed process does all the complicated logic of the application ex. parsing. The privileged process is connected with a sandboxed process over a UNIX domain socket and simply serves the next file descriptors to be parsed. If an attacker exploited our parser (which is very common),


Process in capability mode
Exploited process in capability mode
Privileged process
Privileged process
FS
FS

On the left, the privileged process can open files and serves them to a sandboxed process. On the right, an attacker has exploited the process—access to FS is denied.

he wouldn't have any access to any other part of the system—but only to another read-only file.

Capsicumizing

Capsicumizing is a funny name that we use to describe the process of sandboxing existing applications. Currently in FreeBSD we have around 60 sandboxed applications. From very simple ones such as **yes(1)**, to some more complicated like **jot(1)**, to much more complex ones like **tcpdump(1)** and **ping(8)**. There is a decent number of sandboxed applications, but we still work all the time to sandbox more of them. The full list of applications and current progress can be followed on the Capsicum wiki [6]. In Example 1, we have a patch for capsicumizing **cmp(1)**, a program that compares two files. In this patch, we use a method to pre-open all files that we need before entering capability mode. **cmp(1)** is working on two descriptors **fd1** and **fd2** and both have capability **CAP_FSTAT** and **CAP_FCNTL**, which respectively allow us to get a file status using the **fstat(2)** function and **fcntl(2)** for file control. (The **cmp(1)** uses the **fdopen(3)** function that requires **fcntl(F_GETFL)**. In the original patch, we also limit the amount of **fcntls(2)** using **cap_fctnls_limit(2)**, but this is beyond the scope of this article.) We also give descriptors the **CAP_MMAP_R** capability which allows us to map file into the memory. At the end, we limit the **stdout** descriptor and pre-cache the native language support data (NLS). Finally, we enter the capability mode.

It is very straightforward. For this particular application we didn't even need to reorganize the code because all the files were already opened in one place before parsing, and so the initialization phase was really easy to find. Then we limited some of the descriptors and entered capability mode.

Capsicum Helpers

In Example 1, we have one change that is not obvious when you see it for the first time—calling **catopen(3)** to cache NLS data. Normally when we call the **err(3)** function for the first time, it goes to file system and opens a file for NLS. Unfortunately, in compatibility mode this can't be done because it can't open the file.

There is one more known issue with pre-caching things in **libc** which deals with manipulating time. The functions like **localtime(3)**--when we call them for the first time—pre-cache the current time zone of the machine and they do

```
  --- usr.bin/cmp/cmp.c
+++ usr.bin/cmp/cmp.c
@@ -68,2 +71,5 @@ main(int argc, char *argv[])
        const char *file1, *file2;
+       cap_rights_t rights;
+       unsigned long cmd;
+       uint32_t fcntls;

@@ -148,2 +154,19 @@ main(int argc, char *argv[])

+       cap_rights_init(&rights, CAP_FCNTL, CAP_FSTAT, CAP_MMAP_R);
+       if (cap_rights_limit(fd1, &rights) < 0 && errno != ENOSYS)
+               err(ERR_EXIT, "unable to limit rights for %s", file1);
+       if (cap_rights_limit(fd2, &rights) < 0 && errno != ENOSYS)
+               err(ERR_EXIT, "unable to limit rights for %s", file2);
+
+       cap_rights_init(&rights, CAP_FSTAT, CAP_WRITE, CAP_IOCTL);
+      if (cap_rights_limit(STDOUT_FILENO, &rights) < 0 && errno != ENOSYS)
+               err(ERR_EXIT, "unable to limit rights for stdout");
+
+       /*
+       * Cache NLS data, for strerror, for err(3), before entering capability
+         * mode.
+         */
+       (void)catopen("libc", NL_CAT_LOCALE);
+
+       if (cap_enter() < 0 && errno != ENOSYS)
+               err(ERR_EXIT, "unable to enter capability mode");
+
        if (!special) {
```

Example 1. Capsicumizing `cmp(1)`, simplified patch proposed by Conrad E. Meyer.

that only the first time.

Due to these unclear behaviors, Capsicum helpers were created. Capsicum helpers are a set of small inline functions that allow us to pre-cache things i.e. time zones and NLS data.

The man page for them is also a good place to document this unclear behavior. So for caching time zones and NLS, we will find two functions: `caph_cache_tzdata(3)` and `caph_cache_catpages(3)`.

Another use of Capsicum helpers is to reduce the amount of code needed in our application. If we go back to Example 1, we can see we are limiting `stdout`. The question is how many applications need to limit `stdio`—probably most of them—which is why the `caph_limit_stdin(3)`, `caph_limit_stdout(3)`, `caph_limit_stderr(3)` and `caph_limit_stdio(3)` functions were introduced. Those functions allow us to limit single descriptors with the most common rights, and to limit others with a specific one for the application. If the default limitations are enough for our program, we can simply call the `caph_limit_stdio(3)` function which will limit all of them in a single command.

Another very common pattern is calling the `cap_enter(2)` function, and if the function fails, then checking `ERRNO`. The purpose of this is to check if entering capability mode failed because something happened or if our operating systems just don't support it. In the first case, an application should stop working at this point. In the second case, it should still run because our whole system doesn't support it. At first, this check can be counterintuitive, and it's very uncommon to not check `ERRNO`. This is why we are working on presenting another function `caph_enter(3)` which will hide this check [7].

Debugging Tools

When sandbox is added to an application, a developer cannot notice certain conditions of a program. An application could use a library which a developer doesn't know very well. For example, if an application is using a library and this library uses a random number generator by opening `/dev/random` if possible, otherwise it can use an insecure random generator. If a developer does not notice this behavior by analyzing the code, this can lead to introducing new bugs while sandboxing an application. This is why providing useful debugging tools is one of the biggest challenges in building a sandboxing mechanism. For Capsicum in FreeBSD, we have two tools:

- `ktrace(1)/kdump(1)`,
- gdb with *TRAPCAP*.

During sandboxing, a developer can run a program with `ktrace(1)` and check if no `ECAPMODE` or `ENOTCAPABLE` has been returned. This might cause some issues for a developer, as sometimes it can be hard to know which called function failed. It's also very hard to cover all possible run paths.

In our previous example, this library function which opens `/dev/random` is used with only one particular option provided in the program. However, a program has many options and it could be very easy to miss. This is also why regression tests are so important. Unfortunately, in this case we would need to run a whole test suite under `ktrace(1)` and analyze its output. In Example 2 below, we have a sample output of the `ktrace(1)`.

```
802 random  CALL  cap_enter
802 random  RET   cap_enter 0
802 random  CALL  openat(AT_FDCWD,0x400877,0<O_RDONLY>)
802 random  CAP   restricted VFS lookup
802 random  RET   openat -1 errno 94 Not permitted in capability mode
802 random  CALL  exit(0)
```

Example 2. Result of a `kdump(1)` on a program that enters capability mode and tries to open `/dev/random`.

Another way of analyzing our program is to use a new debugging feature for Capsicum which was implemented by Konstantin Belousov under FreeBSD Foundation sponsorship. Thanks to this, when `ECAPMODE` or `ENOTCAPABLE` is returned, a kernel will issue `SIGTRAP`. As a result, we will get a core dump exactly at the moment the error occurred. This makes it harder to overlook some errors as our program will abort and we will notice it while we run it. A core dump also provides more information about the state of the process when the error occurred. We can enable this feature using `sysctl kern.trap_enotcap` (globally in the system). If there is a need to enable it per-process, we can use `proctl(2)`. For a system to be able to generate a core dump in capability mode, we also need to set `sysctl capmode_coredump`, otherwise programs in sandbox are unable to create them.

Nvlist Library

We've already discussed some methods of obtaining more capabilities in our process. One method is to receive them from another process. To make it easier to split programs between privileged and unprivileged processes, Caspsicum developers also introduced a very easy IPC library—**nvlist**. This library is based on the list containing pairs (name, value), and it allows us to keep many primitives like: numbers, strings, binary and bools.

However, one of the most special things is that it also allows us to keep descriptors on the list. Furthermore, it provides functions to send and receive **nvlist** over the socket. All of this has been designed to allow for separation of processes and for capiscumizing them more easily. **nvlist** as a container also exists in the kernel and is used by some drivers (ex. **ixl**). The implementation used exactly the same code as user land—it doesn't contain primitives that don't exist in the kernel (like descriptor or socket). In Example 3, we can see a simple use of **nvlist**. The author recommends considering **nvlist** as a candidate for a serialization library because the implementation and use of it is very straightforward. If you are interested in the use-cases of **nvlist**, see man page (**nv(9)**) or some external materials [9] [10] [11].

```
nvlist_t *nvl;

nvl = nvlist_create(0);
nvlist_add_string(nvl, "first", "foo");
nvlist_add_number(nvl, "second", 1234);

/*
 * What is also very interesting in nvlist is that we need to check
 * only last operation on nvlist. If one of previous adding would fail
 * we would know that at any point.
 */
if(nvlist_send(sock,nvl)<0){,
        fprintf(stderr, "Unable to send nvlist.\n");
        exit(1);
}
```

Example 3. Simple use of the nvlist. Add two values and send it over the network.

Casper Overview

While splitting programs in the privileged and unprivileged process, we will notice some common patterns. For example, many network tools need to have access to the DNS server. In capability mode, we can't connect to the server directly and need some other process that can talk with it. To simplify this and reduce the amount of the code needed for all applications, the Casper library was created. This library provides us a set of services like:

- **system.dns** - service for getting network host entry
- **system.grp** - service for group database operations
- **system.pwd** - service for password database operations
- **system.random** - service for getting entropy
- **system.sysctl** - service for getting or setting system information
- **system.syslog** - service for syslog

When creating a Casper instance, it forks from the original process and this requires the creation of Casper services (**cap_init(3)**) before entering capability mode. We can also receive a service from another process which has a service. All services are well documented, and in the man pages we can find examples of how to use them.

Currently we are working on one more service—**system.fileargs**. The goal of this service is to provide a simple tool for sandboxing applications, which, as an argument, takes a list of files. This service will provides descriptors via a similar API to **open(2)**. Thanks to the interface, applying it should be straightforward for existing applications. Even though this service is still under development, the project agreed that it will be initially treated as experimental [8]

We also have a plan to implement other services such as:

- **system.login** - a service for accessing the login class capabilities database
- **system.tls** - a service for creating a safe connection using TLS/SSL
- **system.socket** - a service for creating network connections
- **system.configuration** - a service for fetching unified configuration

These are all currently only ideas.

Casper and dhclient(8)

One of the new services is **system.syslog**. Let's discuss for a moment why we created it.

```
Starting devd.
Starting dhclient.
pid 336 (dhclient), uid (65): Path `/var/crash/dhclient.65.0.core'
failed on initial open test, error = 2
pid 336 (dhclient), uid 65: exited on signal 5
Trace/BPT trap/etc/rc.d/dhclient: WARNING: failed to start dhclient
Starting syslogd.
```

Example 4. `dhclient(8)` core dumping during start.

```
void syslog(int priority, const char *message, ...);
void vsyslog(int priority, const char *message, va_list args);
void openlog(const char *ident, int logopt, int facility);
void closelog(void);
int setlogmask(int maskpri);
```

Example 5. `syslog` API

While booting an operating system with **`kern.trap_enotcap sysctl`** enabled, we noticed that **`dhclient(8)`** was core dumping. In Example 4, we present this situation.

After analyzing this program, it turned out that dhclient was using **`syslog`** to report its status. If we look one more time at Example 4, we see that the **`syslogd(8)`** was started after **`dhclient(8)`**. This is the standard problem of the chicken and the egg. **`syslogd(8)`** sometimes needs a network to be configured and **`dhclient(8)`** needs a **`syslogd(8)`** to report status. Historically, we decided to run **`dhclient(8)`** before running **`syslogd(8)`**. What is worth noting is that **`dhclient(8)`** tried to connect to **`syslogd(8)`** before entering capability mode because the server did not exist and yet it failed. Surprisingly, each time the program was not connected, a syslog function tried to connect to it! Of course, because we are now running in Capsicum, we will never be able to establish a connection. So, to solve this issue, we decided to introduce a new Casper service syslog. It tries to connect to **`syslogd(8)`** and if it fails, it will try the next time there is something to report. It's also worth noticing that the syslog API (shown in Example 5) doesn't report any issues if something is wrong.

Summary

Sandboxing base systems is an ongoing process. We have introduced many tools which should lower the entrance barrier for new people wishing to use Capsicum.

Capsicumizing is a very good way to learn about operating systems--how they work, how they interact, and how some libraries behave. There are still many small programs waiting to be capsicumized. This can be a great lesson on operating systems--especially for people who dream about becoming FreeBSD developers. ●

MARIUSZ ZABORSKI is a lead software developer at *Wheel Systems*. He has been a proud owner of the FreeBSD commit bit since 2015. Mariusz's main areas of interest are OS security and low-level programming. At *Wheel Systems*, Mariusz leads a team that is developing the most advanced solution to monitor, record and control traffic in an IT infrastructure. In his free time, he enjoys blogging (http://oshogbo.vexillium.org).

BIBLIOGRAPHY

[1] J. Anderson, *A Comparison of Unix Sandboxing Techniques*, FreeBSD Journal Sept/Oct 2017

[2] P.Dawidek, M.Zaborski, *Sandboxing with Capsicum*, ;login: issue:December 2014, Vol. 39, No. 6

[3] M.Zaborski, *Capsicum and Casper - a fairy tale about solving security problems*, AsiaBSDCon 2016 http://oshogbo.vexillium.org/pdf/AsiaBSDcon2016.pdf

[4] http://en.wikipedia.org/wiki/Namespace

[5] Robert N.M. Watson, Jonathan Anderson, Ben Laurie, Kris Kennaway, *Introducing Capsicum: Practical Capabilities for UNIX, 2010*

[6] FreeBSD wiki, Capsicum page, https://wiki.freebsd.org/Capsicum

[7] Introduce **`caph_enter()`**, FreeBSD phabricator, https://reviews.freebsd.org/D14557

[8] Introduce **`system.fileargs`**, FreeBSD phabricator, https://reviews.freebsd.org/D14407

[9] Introduction to **`nvlist`** part 1, http://oshogbo.vexillium.org/blog/42/

[10] Introduction to **`nvlist`** part 2 - **`dnvlist`**, http://oshogbo.vexillium.org/blog/43/

[11] Introduction to **`nvlist`** part 3 - simple traversing, http://oshogbo.vexillium.org/blog/45/

SEE TEXT ONLY

new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community.

In this installment, the spotlight is on Jason Bacon who received his ports bit in November, Koichiro Iwao who received his ports bit in March, Sean Fagan who received his src bit in April, Vincenzo Maffione who received his src bit in March, Fernando Apesteguía who received his ports commit in March, Tom Jones who received his src bit in April, and Eric Turgeon who received his ports bit in March.

Tell us a bit about yourself, your background, and your interests.

• Jason: I've been into computers since high school, when I took the very first computer class our school ever offered. I was immediately hooked by the stimulating challenges and immediate feedback inherent in computer programming. Onto college, I chose computer science over chemistry in the 11th hour for my undergrad major. Sort of fell into a master's program for logistical reasons immediately after. After teaching computer science full time for a few years, I pursued a PhD program in biology, but was forced to withdraw due to family responsibilities. I'm still a biologist at heart, but my computer science background has served me well and I've managed to find rewarding work as a developer and systems manager.

I now support research computing and high-performance computing at UW Milwaukee, working with researchers in many fields, but mainly engineering and genomics.

I'm a staunch operating system and language agnostic and will always campaign against religiosity in computing. Self-limiting idealism hurts everyone from end-users to sysadmins. I always look openly at alternatives and choose the most efficient tool for the job. Sometimes it's BSD, sometimes Linux, sometimes Windows.

I've also been involved in sports since a young age, including 4 years of tennis and cross country running in high school, and playing tennis for our university team. Since college, I've changed my focus to Chinese martial arts, cycling, XC skiing and sea kayaking. (I live a few blocks from a beautiful stretch of Lake Michigan shoreline park land on Milwaukee's south side.)

Philosophically, the closest label that would fit me is Buddhist/Taoist. I believe in complete openness, destruction of the ego, and that any hint of anger is a sign post pointing directly toward room for personal growth.

• Koichiro: I was born in Kyushu Island located southwest of Japan and I'm still in Kyushu. I am an upstream developer of xrdp which offers Unix-like operating systems a graphical login to a remote client via Microsoft Remote Desktop Protocol. I'm working on an xrdp project on improving FreeBSD compatibility and handling non-ASCII characters such as CJK. I'm also interested in desktop virtualization. You should already know how useful GNU screen and tmux are. Desktop virtualization will provide GUI-like version of GNU screen. You can resume an existing remote session from everywhere over the network.

• Sean: I've been interested in Unix-like systems since I decided to write a PDP-11 simulator in C instead of using an over-loaded RSTS/E system for the college assembly language course I was in.

I went from college to work for The Santa Cruz Operation, and spent most of my time there working on the XENIX port of Microsoft's C Compiler. I played with some kernel aspects (adding a simple form of ACL, just to see if I could), and when SCO came out with their SysVr3.2-based system, which for POSIX-compliance had job-control support, I set about making it work (which also involved porting tcsh from BSD to SysVr3.2 and getting emacs to work with it). I was sending any patches to the open-source code back, and I also started writing some simple, low-level libc routines for the then-new 386BSD. This led to me working for Cygnus, which meant working more directly with open source, although it didn't get that name for a while.

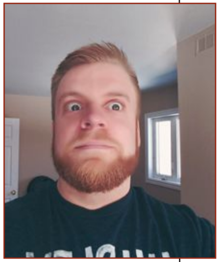
● **Vincenzo:** I'm a Software Engineer and I live in Pisa, Italy. Currently, I'm a last year PhD student at the University of Pisa. My research interests include fast userspace networking, network virtualization, and kernel network drivers. But most of all I love programming and operating systems. In my spare time I also enjoy playing piano, biking around and cooking.

● **Fernando:** My name is Fernando Apesteguía. I'm a software developer living in Madrid (Spain). After I got my Master's Degree in Computer Science, I moved to Holland to work for ESA's YES2 project. It was extremely cool and we were even granted a Guinness World Record for the longest man-made structure ever deployed in space. There I wrote software for the onboard computer running QNX. After that, I went back to Spain and joined Open Sistemas. Here, we use open source software to develop our solutions. I usually do C/C++ programming in Linux using Qt and DBus for IPC.

On the personal side, I live with my girlfriend who suffers from my lack of free time. I enjoy traveling as much as I can and practicing martial arts. I hold a 3rd DAN black belt in karate, which I've been practicing for the last 25 years, and I'm a black belt in other disciplines like SAMBO and Self Defense.

● **Tom:** I am a researcher at the University of Aberdeen in the North East of Scotland, where I work on Internet transport and standardization. For the past few years I have been working on replacing the Socket API with something modern and adaptable in the EU NEAT Project (https://www.neat-project.org/).

Sometimes I have to get out from behind my computer and move it to the local hackerspace, 57North Hacklab. Other times, if the weather is nice, I go further afield and set up my computer in a tent.



● **Eric:** I am the founder of GhostBSD and started to work for iXsystems last September as an Automation Engineer for the QA department. I am married to a wonderful woman named Karine, who is more than patient about me spending time working on GhostBSD, and I have one son named Samuel who turned six last January. Like his father, he likes computers and technology and also is a fanatic of Minecraft. I currently live in Dieppe city in New Brunswick, Canada. I speak French, English, and Chiac (French and English mixed together).

I did not finish high school but I am a curious guy that likes technology and am passionate about Unix, BSD, and desktop software. I learned C, shell script, and Python, but Python became my language of choice for almost everything that does not need to scale like C. I learn everything as I need it, so my knowledge of FreeBSD, Python, C, and even for my new career gets better as time passes.

I like to play with my son and appreciate having quality time with my wife. I love to read history, programming books, and the bible. I love to be involved in the City of Grace church and to have evening parties with our circle of friends. I like mountain biking on crazy trails, and train with PowerBuilder, but lately, I have been lazy with that. I love metal music and playing guitar when I have time.

How did you first learn about FreeBSD and what about FreeBSD interested you?

● **Jason:** I've been involved in Unix development since before FreeBSD existed and have been running both BSD and Linux systems uninterrupted since the mid-1990s. I love FreeBSD's long tradition of solid performance and stability, as well as the devotion to clean and systematic management and software deployment. The FreeBSD ports system has few peers in terms of its size, capabilities, and flexibility. I can set up a FreeBSD system from scratch to serve most purposes in under an hour, knowing that I'll have very little trouble with it from that day on.

● **Koichiro:** I've been using FreeBSD for more than 10 years and remember FreeBSD 5.4 or 6.0 as my first FreeBSD. I started using FreeBSD as network gateways and routers when I was a college student. ipfw has simple syntax and is easy to understand. Also, dummynet is a great tool to control traffic and emulate real-world networks. To sum up, network features interested me in FreeBSD.

● Sean: After the first version of Linux kept panicking when using telnet, I instead switched to 386BSD for my personal computer, and that led to the other BSDs, and, eventually, focusing on FreeBSD a couple of months after it started.

The source-distributable BSDs were still very new at the time, and there was a lot of work that interested me. As has been typical in my life, I did a lot of library and utility work, and some supplemental kernel code. While taking McKusick's BSD internals class, I decided to write a version of procfs, which I did with significant assistance to get the template down. Then once I had procfs working, I started writing truss; these two remain my biggest single contributions. I also wrote some articles for Dr Dobb's Journal related to this.

● Vincenzo: I met FreeBSD for the first time at the university in a course where I learned the basics of UNIX systems administration. However, my first real interaction with FreeBSD source code only happened a couple of years ago, when I participated in a Google Summer of Code project. In this project I had a lot of fun playing with kernel bits and virtualization tools. I also had the chance to present my work at one of the FreeBSD meetings.

I have a strong interest in operating system implementation, and I like FreeBSD in particular because of its clean design and excellent documentation.

● Fernando:The first time I heard about FreeBSD was at high school. A friend of mine was an early Linux user and he told me about FreeBSD. A short time after that, I bought an issue of the Spanish version of PC World which included a copy of FreeBSD 4.2. It didn't work so well with my hardware, but I could install it and play with it for a while. After that I upgraded to 5.0, but I still used Linux more often. I think it was around 7.0 when I became mostly a FreeBSD user. I was impressed by its stability and that feeling has increased over time. I still own a 12-year-old laptop with just 1 GB of RAM running FreeBSD 11.0.

● Tom: I first ran PowerPC FreeBSD on a G4 iBook when I was distro hopping in 2008, but my adventure really started when in 2018 I joined the research group at the University of Aberdeen. At the time the group was working on standardization of a TCP modification called NewCWV (RFC7661). We had an implementation for Linux and the IETF standardization credo is "rough consensus and running code" and it helps if there are implementations for as many platforms as possible.

There was talk of porting to a BSD operating system and I asked (well, maybe I demanded) to be involved. Porting work started in virtual machines on my desktop, but soon I had acquired some second-hand servers to build and run FreeBSD.

FreeBSD was the operating system of choice for the port because of the famously excellent TCP stack. Soon I started to miss the tools that were available on my FreeBSD development machines and decided I had to use FreeBSD as a desktop.

● Eric: II started that journey by being bored with having viruses on Windows XP and I wanted to become a hacker. I switched to PC Linux OS, moved to Mandriva and then to Ubuntu and felt kind of at home. On my quest to become an infamous hacker, I found "How To Become A Hacker" by Eric Steven Raymond around the same time as a life changing event that made me stop my journey to become a hacker. "How To Become A Hacker" talked briefly about BSD Unixes which caught my attention. I started to Google BSD, which started my path to FreeBSD. I installed FreeBSD 7.0 which was easier to install than XP, but the shell was my nemesis. So, I reinstalled Ubuntu, did more research, and found and tried PC-BSD 1.4. I did not like KDE, so I reinstalled Ubuntu on half of the disk, printed the whole FreeBSD Handbook and the docs to install and setup Gnome, and reinstalled FreeBSD on the second half. It is where all my love began for FreeBSD, and I started GhostBSD to have a Gnome equivalent of PC-BSD.

Today my primary interest in FreeBSD is GhostBSD and all my servers and VPS that only run FreeBSD. I am more interested on the Desktop side of FreeBSD than FreeBSD as a server, which is why GhostBSD started.

FreeBSD and Eric Steven Raymond's essay started my path to becoming a programmer, and from all the connections I made on IRC, I found a great career at iXsystems.

How did you end up becoming a committer?

● Jason: I've been maintaining ports since around 2004, when I found that FreeBSD solved a lot of stability issues for us in fMRI research at the Medical College of Wisconsin. I currently maintain dozens of active ports and have a couple hundred more in my work-in-progress collection.

I was invited to become a committer twice. The first time was several years ago during a very hectic period of my life and I reluctantly had to decline. The second time, about a year ago, I was still busy, but things were trending in the right direction, so I bit the bullet and accepted.

I also became a pkgsrc developer around the same time and it has been very interesting to compare the pros and cons of each project. I hope to help bridge the two for the benefit of both in the coming years. We use pkgsrc extensively to install the latest open source packages on our CentOS systems.

• Koichiro: I've been contributing to the FreeBSD Project for years, particularly as a port maintainer of xrdp. One day, I volunteered for a ports committer. Hiroki Sato, a Japanese core developer of FreeBSD, proposed me as a ports committer and finally I became one.

• Sean: I was a committer back near the beginning. However, in 2001, I started working at Apple, which, shall we say, frowned on me continuing that. So, there is a decade-long gap where I hardly ever did anything publicly visible. After Apple, I went to iXsystems, where I was, for the first time, working with FreeBSD full-time. However, most of what I did for the first couple of years involved non-OS code (installer and updater, primarily). Every once in a while, I would come across something I wanted to do that required OS changes, and most of those I emailed back to the last person to touch the files in question. Some patches are still currently at iX, and they need to go back.

In addition, I started porting some ZFSOnLinux changes back to FreeBSD, and these are too large to send as simple patches via email, so with the assistance of Alexander Motin and Kris Moore, I got my commit bit back (still probationally). But I am very much looking forward to getting the ZFS sorted scanning code back soon and finishing up porting the native ZFS crypto code.

• Vincenzo: I'm a maintainer of the open source Netmap project, which offers an API for applications to perform fast network I/O from userspace. Netmap runs on both FreeBSD and Linux, and its source code is currently hosted on GitHub. Although Netmap was already included in the FreeBSD tree, the code in there was not really maintained and was constantly out of date with respect to the upstream version. As a result, during the last AsiaBSDCon I was asked to become a committer to keep the FreeBSD Netmap code in shape and aligned with the upstream. I was very happy to accept.

• Fernando: After becoming a full time FreeBSD user, I started to contribute some port PRs. In 2011, I sent a couple of PRs porting some of the ports listed in the WantedPorts page of the wiki. It was a fun thing to do and it allowed me to learn a lot thanks to the tips given by all the people who committed my patches. I even gave a couple of talks at my Alma Mater (Universidad de Valladolid) about FreeBSD and how to create ports. After some time, I thought it would be good to get more involved in the Project, so I started to end all my PRs with something like "Hey, I would like to step up in case there's someone who can guide me through the mentoring process." After some time, tz@ answered my call. tz@ and tcberner@ are my patient mentors. I'm still happily learning how to become a good ports committer.

• Tom: At the IETF in Prague last year, I was stood up for lunch and was invited to join a colleague and the Netflix crew instead. Over a Reuben sandwich, I got chatting with Jonathan Looney (my mentor) about the work I had done so far on FreeBSD. At BSDCam in Cambridge a few weeks later, Jonathan asked if I was interested in becoming a committer.

It actually took until the March IETF Meeting in London for me to speak to Jonathan and ask what steps I had to do to get the process started. Two weeks later I got the excellent email from Core.

• Eric: II was trying to help Koop on and off with MATE and Gnome when I had time. When I started work at iXsystems, I was a bit more active, and I tried to push MATE 1.20 out in ports, but Koop was not available, so Baptiste offered me a commit bit, and I said yes. Since Baptiste and I are in different time zones, Baptiste caused me to have a second mentor in the same time zone. I mentioned that on our chat at work and William Grzybowski offered to become my co-mentor. I got voted to become a ports committer, and the rest is history to be made.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

• Jason: It's been a struggle to find time to finally master my porting skills, but with the help of my mentor (@jrm) and significant feedback from @mat and a couple others, I'm getting a handle on it and feel that I'm pretty close now. The mentoring program and well-developed tools (poudriere, phabricator, portlint, stage-qa, etc.) really help optimize the learning process.

Don't hesitate to join the team. The time you put into learning from other developers is a great investment that will pay off big in the long run. You'll learn from the vast experience of many who came before you how to solve problems more efficiently and produce solutions that are less likely to break and create more work for you down the road. It's a great opportunity to hone your skills in the company of a worldwide team of some of the best developers alive.

● Koichiro: Nothing has changed so far except I can commit changes by myself. I keep on contributing to the ports tree as before.

If you want to be a committer, keep contributing and create a good track record. It will prove you have enough experience and knowledge when you volunteer to become a FreeBSD developer.

● Sean: Read commit logs, read the mailing lists. Ask questions. Submit code--either patches or new code that should be in the system. Find a mentor. Learn subversion.

● Vincenzo: I've been a committer for less than two months, so I don't have enough experience to give meaningful advice.

However, I feel that FreeBSD is a very friendly community, open to new ideas and new people. This is something I really appreciate, as it is not very common across open source communities. I think the most exciting thing about contributing to open source projects is to see one's own work become part of something used by thousands of users. This is definitely something that attracted me and I think it may be interesting for people to think about becoming a FreeBSD committer.

● Fernando: So far, it's been great! People are really welcoming and nice. The peer review process via Phabricator is really good and something always worth using. The attention to details that all the developers put into their work impresses me. It is not enough if it works. It should work in the best possible way.

About advice: subscribe to the email list that you are interested in (freebsd-ports@ for example). Read all the messages you can and you'll learn a lot. After that, you can send some patches. In the case of the Ports Collection, you can adopt an orphaned port, submit a new port, or just send updates to existing ports. For other subsystems like base, it is always good to have a look at bugzilla's database and find something you can help with. Also, don't be afraid to ask!

● Tom: The community has been amazingly welcoming. For me, joining the project has been a long journey. Writing code is just the first part, getting code accepted into upstream takes as much if not more work. The largest steps forward came from meeting people in person. I met Sevan Janiyan (sevan@) at FOSDEM last year and he invited me to attend BSDCam.

Your first contribution should be something small; there aren't any boogiemen lurking on the mailing lists to shout your diff down. So, fix something, a command, man page or some documentation, and get involved!

● Eric: So far so good. I am pleased with everything I have learned lately and with Baptist, William, and Koop to help I am getting better at creating and maintaining ports. I have released MATE 1.20 in ports. I still have more to learn and everything is happening at a slow pace due to my involvement in a lot of projects.

For anyone who is interested in being involved in FreeBSD, it is simple to get engaged in whatever part of the project interests you. Join the forums, FreeBSD IRC channels, and the mailing list related to your interest. Start to help and work, sending your work to the related people, and sooner or later you will get a commit bit.

DRU LAVIGNE is a doc committer for the FreeBSD Project and Chair of the BSD Certification Group.

MAY, JUNE, & JULY BY DRU LAVIGNE

2018 Events Calendar

The following BSD-related conferences will take place in May, June & July of 2018.

BSDDay Brasil • May 26 • Seropédica, Rio de Janeiro

https://bsdday.com.br/2018/ • This one-day, open source event held at the Universidade Federal Rural do Rio de Janeiro features several FreeBSD presentations. This event is free to attend and the presentations will be in Portuguese.

BSDCan • June 6–9 • Ottawa, Canada

http://www.bsdcan.org/2018/ • The 15th annual BSDCan provides 2 days of tutorials, a Developer Summit, and 2 days of technical talks that appeal to a wide range of people from extreme novices to advanced developers. Registration is required for this event.

USENIX Annual Technical Conference • July 11–13 • Boston, MA

https://www.usemix.org/conference/atc18 • The 2018 USENIX Annual Technical Conference will bring together leading systems researchers for cutting-edge systems research and unlimited opportunities to gain insight into a variety of must-know topics.

OSCON 2018 • July 16–19 • Portland, OR

http://conferences.oreilly.com/oscon/oscon-or/ • OSCON brings together talented people from diverse backgrounds who are doing amazing things with open source technologies. Explore the latest tools and technologies, get expert in-depth training in crucial languages, frameworks, and best practices, and get exposure to the open source stack in all possible configurations.

svn UPDATE

by Steven Kreuzer

The 11.2-RELEASE code slush is currently in effect, and by the time you read this, there should be a releng/11.2. Expect to see a release announcement around June 27, 2018 and start preparing to upgrade, because FreeBSD 11.1 will reach end of life on September 30, 2018.

Introduce dwatch(1) as a tool for making DTrace more useful—

https://svnweb.freebsd.org/changeset/base/330559

dwatch(1) is a tool for making dtrace more useful. dwatch provides a fun and painless way to do everything from watching the system process scheduler in realtime, to filtering out filesystem events and everything in between.

Add support for zstd-compressed user and kernel core dumps—

https://svnweb.freebsd.org/changeset/base/329240

This works similarly to the existing gzip compression support, but zstd is typically faster and gives better compression ratios. Support for this functionality must be configured by adding ZSTDIO to one's kernel configuration file. dumpon(8)'s new -Z option is used to configure zstd compression for kernel dumps. savecore(8) now recognizes and saves zstd-compressed kernel dumps with a .zst extension.

Read-behind / read-ahead support for zfs_getpages()—

https://svnweb.freebsd.org/changeset/base/329363

ZFS caches blocks it reads in its ARC, so in general, the optional pages are not as useful as with filesystems that read the data directly into the target pages. But the optional pages are still useful to reduce the number of page faults and associated VM / VFS / ZFS calls. Another case that gets optimized (as a side effect) is paging in from a hole. ZFS DMU does not currently provide a convenient API to check for a hole. Instead, it creates a temporary zero-filled block and allows accessing it as if it were a normal data block. Getting multiple pages one by one from a hole results in repeated creation and destruction of the temporary block (and an associated ARC header).

Reduce ARC fragmentation threshold—

https://svnweb.freebsd.org/changeset/base/315449

As ZFS can request up to SPA_MAXBLOCKSIZE memory block e.g. during zfs recv update, the threshold at which we start aggressive reclamation to use SPA_MAXBLOCKSIZE (16M) instead of the lower zfs_max_recordsize which defaults to 1M.

Fix OpenDowngrade for NFSv4.1 if a client sets the OPEN_SHARE_ACCESS_WANT* bits—

https://svnweb.freebsd.org/changeset/base/332790

The NFSv4.1 RFC specifies that the OPEN_SHARE_ACCESS_WANT bits can be set in the OpenDowngrade share_access argument and are basically ignored. It also changes the error from NFSERR_BADXDR to NFSERR_INVAL since the NFSv4.1 RFC specifies this as the error to be returned if bogus bits are set. (The NFSv4.0 RFC didn't specify any error for this, so the error reply can be changed for NFSv4.0 as well.)

Make lagg creation more fault tolerant—

https://svnweb.freebsd.org/changeset/base/332645

Warn, don't exit, when SIOCSLAGGPORT returns an error. When we exit with an error during lagg creation, a single, failed NIC (which no longer attaches) can prevent lagg creation and other configuration, such as adding an IPv4 address, and thus leave a machine unreachable. Preserve non-EEXISTS errors for exit status from SIOCSLAGGPORT in case scripts are looking for it. Hopefully, this can be extended if other parts of ifconfig can allow a "soft" failure. Improve the warning message to mention what lagg and what member are problematic.

Add support for TCP high precision timer system (tcp_hpts)—

https://svnweb.freebsd.org/changeset/base/332770

It is the forerunner/foundational work of bringing in both Rack and BBR which use hpts for pacing out packets. The feature is optional and requires the TCPHPTS option to be enabled before the feature

svn **UPDATE** continued

will be active. TCP modules that use it must assure that the base component is compiled in the kernel in which they are loaded.

Remove caching from getlogin(2)—
https://svnweb.freebsd.org/changeset/base/332119

This caching has existed since the CSRG import but serves no obvious purpose. Sure, setlogin() is called rarely, but calls to getlogin() should also be infrequent. The required invalidation was not implemented on aarch64, arm, mips, amd riscv so updates would never occur if getlogin() was called before setlogin().

Remove support for the Arcnet protocol—
https://svnweb.freebsd.org/changeset/base/332490

While Arcnet has some continued deployment in industrial controls, the lack of drivers for any of the PCI, USB, or PCIe NICs on the market suggests such users aren't running FreeBSD. Evidence in the PR database suggests that the cm(4) driver (our sole Arcnet NIC) was broken in 5.0 and has not worked since.

Add RFC 5424 syslog message output to syslogd—
https://svnweb.freebsd.org/changeset/base/332510

Add fprintlog_rfc5424() to emit RFC 5424 formatted log entries. Add a "-O" command line option to enable RFC 5424 formatting. It would have been nicer if we supported "-o rfc5424", just like on NetBSD. Unfortunately, the "-o" flag is already used for a different purpose on FreeBSD. For people interested in using this, the feature can be enabled by adding the following line to /etc/rc.conf:syslogd_flags="-s -O rfc5424"

Add sortbench—
https://svnweb.freebsd.org/changeset/base/332796

This is a set of benchmarks of qsort, mergesort, heapsort, and optionally wikisort and a script to run them.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

SEE TEXT ONLY

conference REPORT

AsiaBSDCon 2018

This was my first time participating in a major international conference and the experience truly exceeded my expectations. I arrived in Tokyo a day before the Paper Sessions started because I also wanted to attend the bhyvecon meeting. bhyvecon was about the FreeBSD hypervisor, bhyve, the subject of the paper I later presented. At the main AsiaBSDCon conference I was very excited to have an opportunity to talk with other attendees about my work porting bhyve to the ARMv8-A architecture. The project is ongoing and the questions I received helped me more clearly outline my immediate goals for the project.

The keynote presentation was a highlight. I like kernel programming, and after attending the talk I was able to attach a clearer meaning and purpose to anykernels and unikernels.

The people I met at AsiaBSDCon were all passionate about their various projects, I enjoyed the conference a lot, and I can't wait to participate again next year.

Alexandru Elisei is studying for his Bachelor's Degree in Computer Science at University Politehnica of Bucharest. He is very passionate about computers and open source software and he has made contributions to various open source projects, including Moodle and Gentoo.

AsiaBSDCon 2018 was my first BSD conference and it was amazing. I met a lot of people from the BSD community and learned of many interesting projects.

The second day after I arrived in Tokyo, I participated at bhyvecon, a conference where BSD users talk about bhyve, FreeBSD's hypervisor. Since the project I'm working is related to bhyve, I knew that would be a great opportunity to find out new things and to meet and talk with bhyve users. It was, indeed, an interesting experience.

I attended both paper presentation days. It was amazing to see so many people developing such interesting features for BSD products and talking about new features and project ideas. I had the opportunity to present the work I'd done for the "Save and Restore feature for bhyve" project, an ongoing project at University POLITEHNICA of Bucharest. This particular feature allows you to snapshot a bhyve guest and to restore that saved state in the future—an important first step towards a live migration feature. At AsiaBSDCon2018 I presented two new features from the "Save and Restore feature for bhyve" project. The first one is related to saving and restoring virtual devices' states, and the second one is related to saving and restoring the virtual machine's CPU state for AMD CPUs.

I found it very interesting that there were projects represented at the conference that used BSD-like operating systems and that you could interact with their developers. A project I liked very much was one that used FreeBSD on RaspberryPi to control a toy car.

Elena Mihailescu is currently pursuing a Master's degree in Security of Complex Networks at The Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest. Her domains of interest include operating systems internals and computer security. She started working on FreeBSD virtualization in September 2017 when she began implementing a Save and Restore feature for bhyve for AMD CPUs.